TAMPERE UNIVERSITY OF TECHNOLOGY

MENGNAN WANG

COMPARISON BETWEEN DROPTAIL AND AQM-RED IN WIRELESS NETWORK

Master's thesis

Examiner: Yevgeni Koucheryavy,
Dmitri Moltchanov
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 9 May 2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
MENGNAN WANG: Comparison between Droptail and AQM-RED in Wireless Network
Master of Science Thesis, 50 pages
May 2012
Major subject: Communications engineering
Examiner: Professor Yevgeni Koucheryavy, Dr. Dmitri Moltchanov
Keywords: Droptail, RED, queue management, congestion control, congestion avoidance, throughput

Queue management is a way to control congestion. There are two algorithms used with TCP protocol that are commonly utilized in nowadays network: Droptail and RED. Droptail is easy to implement but has the problems of lockout and synchronization; RED is complicated but it avoids congestion in advance and reduces the average queue size. Most researches concentrate on the performance of the two algorithms in wired network. But in this thesis, we discuss the performance, i.e. throughput, in wireless network.

The results indicate that RED has no superiority compared to Droptail with different amount of nodes and packet loss probability in wireless network. In the future, researchers can optimize RED or develop other algorithms fit for wireless network.

The thesis first introduces the background of congestion control, the basic knowledge of Droptail and RED, and the related techniques that we used in the simulation. Then the two simulation models based on NS2 is presented. Details of implementations are given as well. The results of comparison are in the last part of the thesis.

PREFACE

This Master of Science Thesis, Comparison between Droptail and AQM-RED in Wireless Network, has been carried out in the Department of Communications Engineering at Tampere University of Technology, Finland. The works has been done during the year 2011-2012 at the Institute of Communication Engineering, Tampere University of Technology.

I would like to say thank you to Yevgeni Koucheryavy and Dmitri Moltchanov for teaching and guiding me with this thesis, and encouraging me when I got trouble and felt upset. Thanks to my friend in Finland for sharing their experiences of doing researches and academic work. Finally, I feel grateful to my parents, who support me all the time.

Tampere, May 2012

Mengnan Wang
mengnan.wang@tut.fi
Orivedenkatu 8 A 22,
33720, Tampere,
Finland
Tel. +358 404411194

CONTENT

ABBREVIATIONS AND DEFINITIONS

ACK	A flag used in TCP to acknowledge receipt of a packet
AGT	Agent trace in trace file
AQM	Active Queue Management
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
cwnd	Congestion window
Droptail	Drop from tail
Headdrop	Drop from head
NS2	Network Simulator 2
RED	Random Early Detection
RTR	Router trace in trace file
ssthresh	Slow start threshold
TCP	Transmission Control Protocol
WIFI	Wireless Fidelity

1. INTRODUCTION

1.1 Background

Congestion is a problem that happens in networking when there is so much data that the network cannot burden anymore. It has a huge influence to both wired network and wireless network and causes the problems of packet loss, packet delay and lockout. A long time could be taken to recover from that situation.

To control congestion, several techniques are used, such as exponential backoff, congestion control in TCP, priority schemes, and queue management. Exponential backoff is used in CSMA/CA, which is the sensing scheme of 802.11. The sender senses the channel before transmission. If the channel is busy, it waits until idle and sends the data after a random period. The random period is calculated by exponential backoff.

Congestion control in TCP consists of slow start, congestion avoidance, fast retransmit and fast recovery [7]. It is a method to control the transmitting rate of the sender. The TCP flow starts at a very low rate and increases exponentially to a threshold. Congestion avoidance then happens and the congestion window increases by one segment each time for one successful transmission. Fast retransmit is first used in TCP Tahoe to retransmit lost packet. And fast recovery is first appeared in TCP Reno after the step of fast retransmit to skip the slow start. Details of TCP Tahoe and TCP Reno will be given in the following chapter.

Priority scheme marks packets into different priorities and drops low priority packets when it is needed. It is not a real congestion control method but improves the performance with other methods.

Queue management is a way to control the queue size of the bottlenecks. It contains passive queue management, which drops packet when the queue is full; and active queue management, which drops packet before the buffer getting full. Droptail and Random Early Detection (RED) are algorithms that represent the two ways respectively. RED is more complicated but can avoid congestion and lockout. The two algorithms are both used in nowadays network [4] [5] [6].

Most of the researches of the two algorithms are in wired network. And RED is used in most of the real wired networks. However, as WIFI and mobile techniques become

popular in people's lives, we want to compare the performance of RED and Droptail in wireless network.

Wireless network is quite different from wired network. The nodes have no wired links to other nodes and base stations, so any barrier that is between the sender and the receiver could influence the transmission power and bring packet loss. Those packet losses are called external sources of losses. The data suffers large-scale fading, which is brought by buildings and large shadows; small-scale fading, which is influenced by multipath; and collision, which happens when several packets arrive to a bottleneck at the same time. All the three aspects cause packet loss. Our goal is to compare RED and Droptail in wireless network with such external sources of losses, and find out whether RED should be used, because RED is more complex than Droptail.

To compare the performance of the two algorithms, simulation models should be established. We set up two models for this thesis; the details are presented in chapter three. In the models, both exponential backoff and TCP Reno are used just as the real wireless network. Priority scheme is not considered in our thesis because the normal RED is our choice for this simulation. The performances we compared include the average queue size and the throughput with different probability of external sources of losses.

1.2 Introduction of chapters

The second chapter introduces the basic knowledge of different queue management algorithms. We first compare the advantages and disadvantages of passive queue management and active queue management, and then give the methods of Droptail and RED. The idea of TCP Reno, which we used in the later simulation, is also presented in this chapter.

Chapter three is a chapter concentrates on the simulation models. We set up two simulation models for this thesis. One is the abstract model, which is used to get the relationship between packet loss probability and throughput, and the relationship between packet loss probability and average queue length. The other is the 802.11 model, which is to obtain the relationship between packet loss probability and the amount of nodes. Both models are simulated on NS2 platform. We analyse the details of the codes, extend the knowledge to CSMA/CA and exponential backoff, and obtain data from the trace files.

The fourth chapter presents all the results we obtained from the simulation models. The results have been counted statistically to make clear graphs. Confidence interval is

added to the results of related to throughput as well. All important parameters that we used are given above the graphs.

The last chapter is the conclusion chapter. It mentions what we have done for this thesis and makes the conclusion of the results.

The codes are attached in appendix, followed by references.

2. QUEUE MANAGEMENT ALGORITHMS

2.1 Introduction of queue management

2.1.1 Congestion control

Queue management is used to control and optimize queues. In networking, it is needed when several nodes transmit data to a bottleneck link. The following figure is a basic topology of network. The bandwidth between sources and Router 1 is 10 Mb, but the bandwidth between routers is 2 Mb. So the link between routers is the bottleneck link. When packets from Source 1 and Source 2 arrive to Router1, they queue and wait for transmission. However, the buffer size is limited. If the buffer is full and the sources keep transmitting, congestion will occur. It will cause congestion collapse and lockout, and the network will restore after a long time [1].

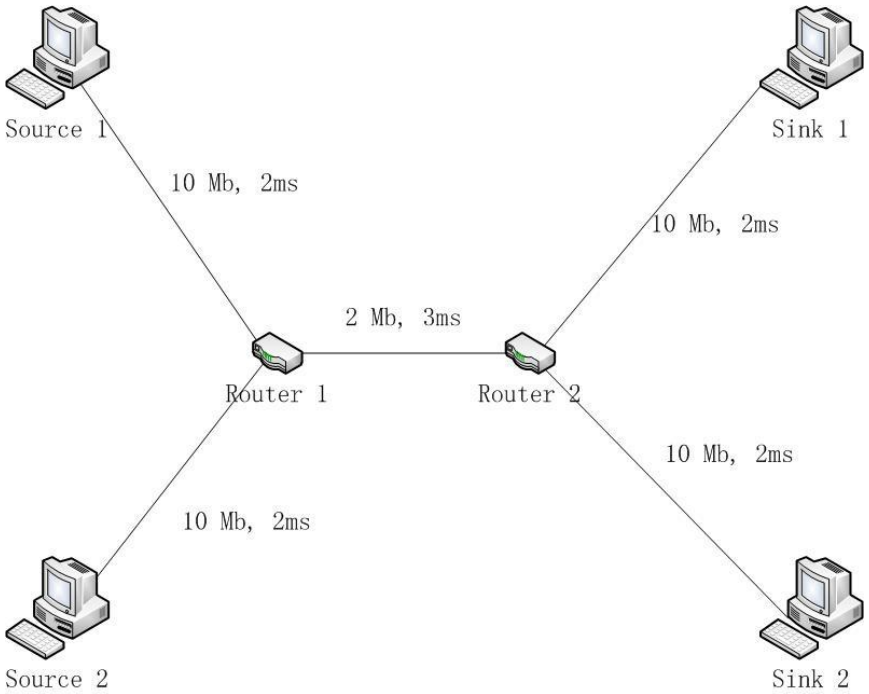

Figure 2.1. *Basic topology of network*

Queue management is one of the methods to control congestion. It controls the queue length of the buffer when certain condition is reached. Besides that, CSMA/CA

(Carrier Sense Multiple Access with Collision Avoidance) with exponential backoff and TCP congestion control are used in wireless networks. Those methods are all considered in the thesis.

2.1.2 Active vs. passive queue management algorithms

The buffer size cannot be too large to increase the packet delay, but we can control the queue size by queue management algorithms. In this thesis, we only consider the algorithms related to TCP.

There are two basic methods of queue management [25]. One is called passive queue management. In this method, packets are dropped only if the buffer is full. It contains several algorithms, such as Droptail, Headdrop, and Push out [2] [3].

Droptail is the most common algorithm of passive queue management. It drops packet from the tail of the buffer when the queue is full, and does nothing when the buffer still has space. Headdrop drops packets from the front of the queue. The delay time is less than that of Droptail because it drops old packets and keeps new packets. Push out is an algorithm that pushes the last queued packet out and puts the coming packet in if the buffer is full.

Passive queue management algorithms are easy to implement in real networks. However, some disadvantages need to be mentioned. First of all, the average queue length of passive queue management algorithms is large for a long period time. As a result of that, the end-to-end delay is long. Second, passive queue management is a way of congestion control, but not a way of congestion avoidance. The sources reduce the rate of transmitting when the queue is full. Those packets that transmitted before the reduction are all dropped and need to be retransmitted. Moreover, passive queue management algorithms cause the problem of lock out. In this case, a single connection transmits normally while others decrease their rates. Then the buffer is full of the packets from that connection. The fairness cannot be guaranteed.

To avoid congestion and lock out, active queue management algorithms are proposed. Those methods begin to control the queue length before the buffer getting full. The algorithm that widely used is RED [4] [5]. In this algorithm, packet is dropped with a given probability when the average queue size achieves the threshold; and the probability increases when the average queue length becomes long. By doing that, the sender cannot get the acknowledgement of the dropped packet from the receiver, and will reduce the transmission rate and adjust the cwnd of TCP protocol before the queue getting full.

Adaptive RED is an algorithm that uses RED with different maximum dropping probabilities depending on the arriving rate; and weighted RED is another RED that

changes the maximum threshold and minimum threshold according to the priority of a packet. Those two active queue management algorithms are better than RED, but are also more complicated to implement.

Active queue management algorithms reduce the average queue size of the buffer. The end-to-end delay is decreased as well. However, all of the active queue management algorithms are more complicated than passive queue management algorithms. So choosing queue management algorithm is important for networks. In following sections, we will discuss the advantages and disadvantages of Droptail and RED, the two main algorithms of passive queue management and active queue management respectively, and simulate the performance of them in wireless network using NS2, to find out the suitable algorithm for TCP transmission.

2.2 Droptail

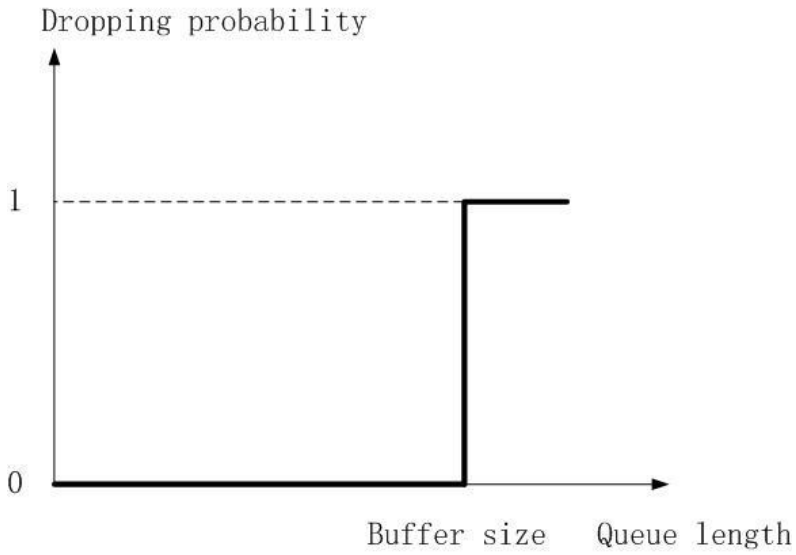

Figure 2.2. *Dropping probability of Droptail*

Droptail is the common algorithm of passive queue management. It drops all the new packets when the buffer is full, and does nothing when the buffer still has space [3] [6].

The figure above shows the dropping probability of packets. The only two dropping probabilities are 0 and 1. When the number of packets arrived to the queue larger than the buffer size, the probability of packet dropping is 1. Otherwise the dropping probability is 0.

The algorithm is easy to implement and does not have complicated parameters. But it also gets the problems of lock out and synchronization. In lock out situation, most of

the connections decrease their transmitting rates except one or few of them. Then the buffer is full of packets from connections with high rates. Synchronization is different. Nearly all connections increase or decrease their transmission rates together. The packets will have a huge delay and loss in this situation. The average queue length is always high if there are a large number of sources.

2.3 RED

RED is an algorithm of active queue management and it is developed for TCP only. It starts to drop packets when the average queue size is larger than the minimum threshold, and changes the dropping probability to 1 when the average queue size is larger than the maximum threshold. The dropping probability varies from 0 to p when the average queue length is between the two thresholds [3] [4]. The figure below presents the dropping probability of RED (Droptail in red line):

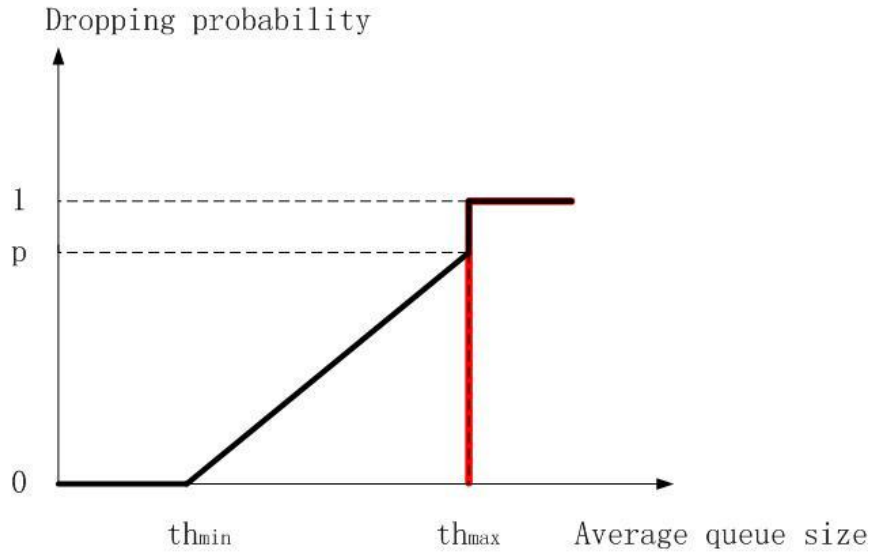

Figure 2.3. *Dropping probability of RED compared to Droptail*

One difference between RED and Droptail is that RED uses average queue length instead of instant queue length. The average queue length is calculated by a low pass filter. This is to consider the transmission rate and reduce the influence of bursty traffic. The equation is:

$$a = (1 - w) * a + w * q \qquad 2.1$$

where a is the average queue size, q is the instant queue size, and w is the weight of the queue.

The dropping probability of RED varies depending on the max probability, minimum threshold and maximum threshold. Once the average queue reaches the minimum threshold, the dropping probability is:

$$\mathrm{p} = p_{max} * \frac{a - T_{min}}{T_{max} - T_{min}} \qquad 2.2$$

where p is the dropping probability, p_{max} is the maximum dropping probability, a is the average queue size, T_{max} and T_{min} are the maximum threshold and minimum threshold respectively.

Once an arrived packet is marked as the dropping packet, the probability of packet dropping changes immediately. The dropping probability of the coming packet is:

$$\mathrm{p} = \frac{\mathrm{p}}{1 - \mathrm{N} * \mathrm{p}} \qquad 2.3$$

where N is the number of packets that has been marked to drop.

The setting of maximum threshold and minimum threshold should be considered in different networks. The size of minimum threshold relates to the purpose of the network. Small minimum threshold results in small delay, while high minimum threshold leads to high link utilization. Typically, maximum threshold is two times greater than minimum threshold.

It is clear that RED is complicated and the performance of it depends on the parameters we choose. But it solves the problem of lock out and synchronization, and avoids congestion before the buffer getting full. In chapter 3, we will set up the system model to compare the two algorithms in wireless network.

2.4 TCP congestion control

As we concentrate on the queue management algorithms with TCP protocol, we should understand TCP congestion control as well. Besides the control of queue, TCP protocol starts to control congestion from transmitting rate. The processes are: slow start, congestion avoidance, fast retransmit and fast recovery [8] [15] [23] [24].

- Slow start

The TCP protocol starts with a very low rate to ensure the success of transmission. Then the rate grows exponentially to reach the slow start threshold (ssthresh). The process ensures that the bandwidth is fully utilized.

- Congestion avoidance

After reach the ssthresh, the rate increases linearly, i.e. one full-sized segment each time, until one packet lost. The process avoids bursty traffic and large amount of packet loss.

- Fast retransmit

Receiver sends duplicate ACKs with the same segment number if one packet has lost and the sender keeps transmitting. If the sender receives three duplicate ACKs, it will retransmit the packet without waiting for the RTT. Fast retransmit is applied in both TCP Tahoe and TCP Reno.

- Fast recovery

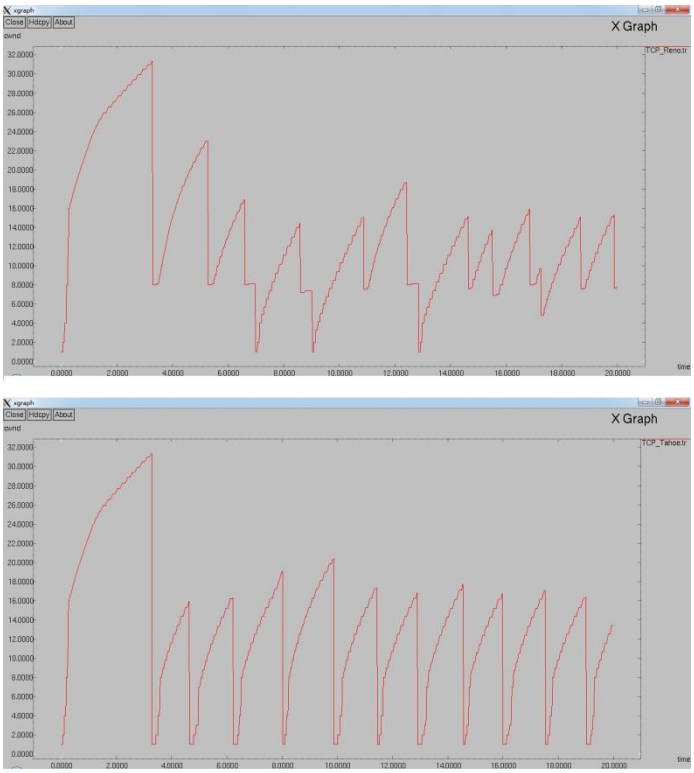

Figure 2.4. *Comparison of TCP Reno and TCP Tahoe*

Fast recovery is first used in TCP Reno. Unlike the ssthresh of TCP Tahoe, which restarts from one every time, the ssthresh of TCP Reno restarts from: max(Flightsize/2,2*MSS) as showed in figure 2.4. And the congestion window (cwnd) equals to ssthresh+3. It is obvious that TCP Reno gains a better performance than TCP Tahoe.

In the later simulation model, we will use TCP Reno as the TCP protocol.

2.5 CSMA/CA and exponential backoff

Collisions may happen at the base station if two messages arrive in the same time slot. The more the amount of nodes is, the more the number of collisions is. In 802.11 network, CSMA/CA is used to prevent collisions. The processes are showed as figure 2.5.

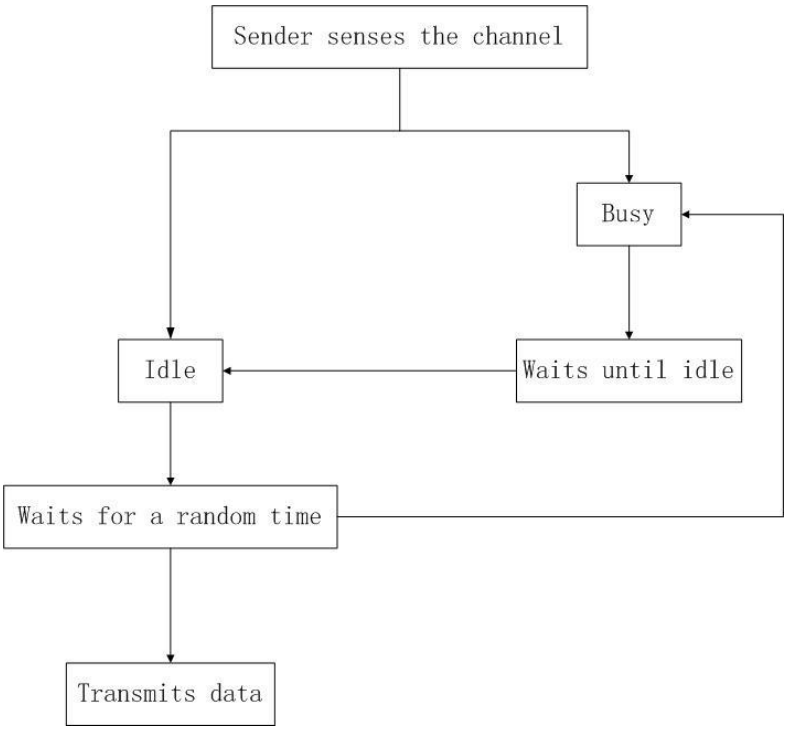

Figure 2.5. *Process of CSMA/CA*

Before transmitting, the sender senses the channel and determines whether the medium is idle or not. If the channel is idle, the sender counts a randomly time using exponential backoff and then transmits the data. Otherwise it waits until the channel becomes idle, and sends the data after a randomly chosen duration counted by exponential backoff, before attempting the transmitting again [9] [20] [21]. Although CSMA/CA reduces the probability of collisions, they still happen during simulation.

RTS/CTS handshake is another technique used in CSMA/CA. before transmitting data, the sender should transmit RTS (Request to Send) to the receiver. And the receiver sends CTS (Clear to Send) back to the sender. The overloads of RTS and CTS packets are small. This reduces the overload of invalid transmission. RTS/CTS also avoids the problem of hidden nodes. The process of handshake can be observed in the trace files of NS2.

In CSMA/CA, exponential backoff is used to generate the random duration. This is to protect the channel from a bursty traffic when several senders sense the idle channel and transmit together. Thus, exponential backoff reduces the probability of collision when there is a large amount of senders.

Exponential backoff delay is an integer multiple of time slot. The number of slots to delay depends on a uniformly distributed random integer r as [10] [22]:

$$0 \leq \mathrm{r} < 2^k, k = \min(n, 10) \qquad 2.4$$

where n is the number of retransmission attempts.

The sender stops the counting of backoff time if the channel becomes busy during this period. If all the attempts are failed, an error report is submitted.

3. SYSTEM MODEL

3.1 Introduction and purpose

The system models are simulated using NS2, a discrete event network simulator [11] [12] [28] [31]. With this simulator, the topologies of the models are clearly presented. And the processes of packet transmitting and packet dropping are reflected. By analysing the trace files and monitoring the TCP sinks, we can easily get the information of the network in details, such as ‘a packet is sent at 0.1 second’. Those details will help us to analyse the performances of RED and Droptail in wireless network.

To find out the performances of RED and Droptail in 802.11, we need to test the throughput and average queue size of the two algorithms with external source of losses. So the first model is an abstract model that simulates wireless packet losses in a wired network. This model is to obtain the relationship between throughput and probability of packet loss with a given amount of nodes.

Next, we simulate the 802.11 model. In this model, collision happens when packets from senders arrive to base station even CSMA/CA with Backoff algorithm is used. The simulation is to gain the connection between number of nodes and probability of packet loss in a wireless model. With those two models, the relationship among throughput, packet loss probability and number of nodes is obtained.

3.2 Abstract model

3.2.1 Topology

The abstract model simulates wireless packet losses in a wired network. The purpose of the model is to find the relationship between throughput and probability of packet loss.

The topology of the abstract model is showed in Figure 3.1:

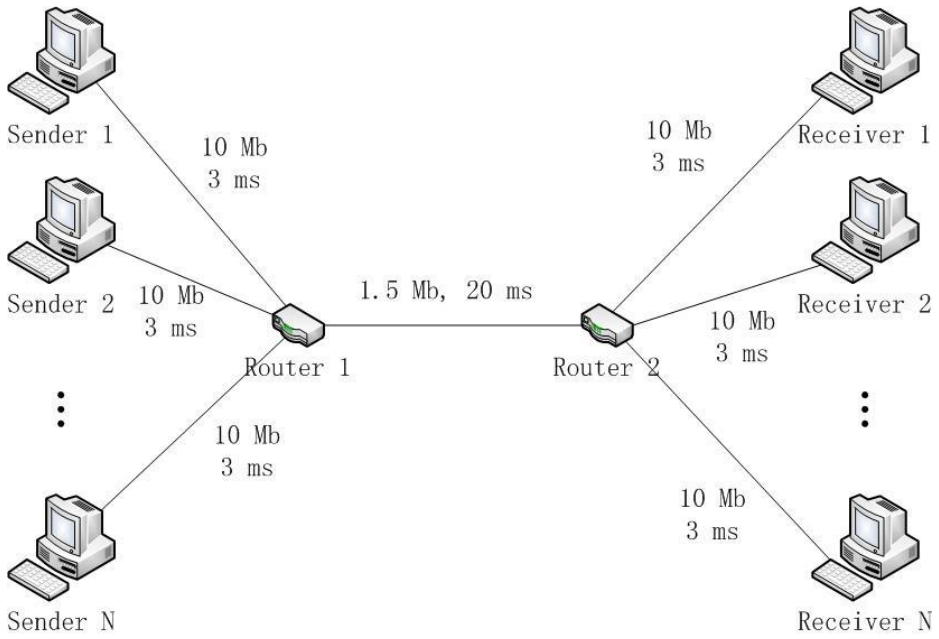

Figure 3.1. *Topology of the abstract model*

The network consists of senders, receivers and two routers. The number of senders or receivers can be set to 2, 5, 10, 20 or 40. Each sender transmits messages to the receiver which has the same number to it through the two routers. The bandwidth of the link between router and node is 10 Mbps, and the delay is 3 ms. The link between two routers has a bandwidth of 1.5 Mbps and a delay of 20 ms.

It is clear that the link between routers is the bottleneck link of the network. The packets sent from the sources queue in the buffer of Router 1 and wait for transmitting. If the queue becomes full and the senders keep transmitting, congestion will happen. To manage the queue and control congestion, the two algorithms: RED and Droptail are used.

3.2.2 Droptail and RED

In Droptail algorithm, new packets are dropped when the buffer of Router 1 is full to control congestion. In this way, if the sender cannot receive the acknowledgement of a packet after an RTT timeout, it will slow start to limit the total number of unacknowledged packet on the link. The ssthresh and cwnd are calculated as TCP Reno algorithm.

However, in RED algorithm, packets may be dropped before the queue is full to prevent congestion. The probability of packet dropping varies due to the average queue size, which is calculated as showed in section 2.4. When the sender does not receive the acknowledgement of a packet after an RTT timeout, it slow starts to control the number of unacknowledged packet on the link, same to Droptail. Several important parameters

such as the minimum threshold, maximum threshold and queue weight should be set before simulating [16] [17] [26] [32]:

- byte_

byte_ is set to true for byte mode, set to false for packet mode. In this thesis, we choose packet mode.

- queue_in_bytes_

queue_in_bytes_ is set to true for queue in byte mode, and set to false for queue in packet mode. In this thesis, we choose packet mode.

- thresh_

thresh_ is the minimum threshold of packet dropping. Packets may be dropped when the average queue size is larger than thresh_. A small thresh_ leads to small delay, while a large thresh_ leads to high link utilization.

- maxthresh_

maxthresh_ is the maximum threshold of packet dropping. Dropping probability is set to 1 is the average queue size is larger than maxthresh_. It is always at least two times larger than thresh_.

- linterm_

linterm_ controls the dropping probability of RED. The dropping probability varies from 0 to 1/ linterm_ if the average queue size is between thresh_ and maxthresh_. It also influences the link utilization.

- q_weight_

q_weight_ is actually the parameter ω in section 2.4. It is the weight to calculate the average queue size.

- drop_tail_

This is a bool parameter. If it is set to true, packet is dropped from tail. RED also contains the parameter of drop_front_ and drop_rand_. But we only set drop_tail_ to true in this thesis.

RED parameters are much more than those. The default value of the parameters can be found in ns-default.tcl in any version of NS2.

3.2.3 Error model

Error model is added to the model to simulate the external source of losses [13] [29]. The rate of loss can be set from 0 to 1.

- rate_

rate_ is the dropping rate of error model.

- unit

unit of error model refers to the mode of the model. Generally it is set to packet mode, i.e. pkt.

- ranvar

ranvar refers to the random variable of error model. Generally it is set to uniform mode.

- drop-target

drop-target is important when there are several types of packets. However, in our simulation, there is only one type of packets. So this parameter is set to Null.

We choose the random variable to be uniform distributed in this model. The loss in the channel can be easily distinguished from the trace file, although there are losses in both the buffer and the channel. The following data is the external loss information in the trace file:

```
- 0.316432 0 10 tcp 1500 ------- 0 0.0 5.0 18 97
d 0.316755 10 11 tcp 1500 ------- 1 1.0 6.0 8 52
```

Before the dropping information, it is not the receive data of that packet at the router, which means the packet dropped is not just after the packet is received by the router.

And the packet dropped because of queue management is presented below:

```
+ 0.062648 80 81 tcp 1500 ------- 22 22.0 62.0 2 125
d 0.062648 80 81 tcp 1500 ------- 22 22.0 62.0 2 125
```

The packet is dropped just after it has been received by the router. That is the difference between the two types of drops.

Although packet delay is a reason of slow start and retransmitting, we do not consider that in our system, because compared to packet loss, the influence of packet delay is quite slight.

3.2.4 Throughput

The throughput of the receiver is obtained by monitoring the TCP sink [30]. The expression of throughput in Mbps is:

$$\mathrm{T} = \mathrm{B} * \frac{8}{t * 10^6} \qquad 3.1$$

where T is the throughput, B is the amount of bytes that has been received, and t is the simulation time.

In this simulation, we record the average throughput between Sender 1 and Receiver 1 and write it in a trace file. The interval of each sample is 0.2 s. the layout of the trace file is:

60.000000000000313 0.27120533333333191
60.200000000000315 0.272497009966776
60.400000000000318 0.27278675496688598
60.600000000000321 0.2742627062706256
60.800000000000324 0.27355789473684067

The first row is the simulation time and the second row is the average throughput from the beginning to that moment.

Figure 3.2. *Throughput of Droptail and RED with 5 nodes and 0.2 packet loss prob.*

The above figure presents the throughput of Droptail and RED. Those graphs are printed by analysing the trace file. The green lines are instant throughput and the red

lines are average throughput. With the graphs, we can clearly see how the throughput changes. The data of throughput is statistically counted and summarized in the next chapter.

3.2.5 Average queue size

The average queue size of the buffer is got by an awk code which analyses the trace file line by line. The information in a trace file is as following [18]:

```
+ 0.020288 0 3 tcp 1500 ------- 1 0.0 5.0 1 6
- 0.020288 0 3 tcp 1500 ------- 1 0.0 5.0 1 6
+ 0.020288 0 3 tcp 1500 ------- 1 0.0 5.0 2 7
```

The information we need is in row 1, 2, 3, 4, 6, and 12:

- Action

The first row is action row that indicates if the packet is arrived, departed, received or dropped. It has four symbols: '+' (arrived), '-' (departed), 'r' (received), and 'd' (dropped).

- Time

The second row presents the time that the action happens.

- Source node

The node that transmits that data is showed in row 3. '10' is the node address of the source node. It has to be clear that the source node is not always the real sender of that data. It could be the router or any forwarder on the link.

- Destination node

The source node is followed by the destination node in row 4. Like the source node, this is not always the real receiver of the data. The destination node could be any node that forwards this data to the receiver.

- Packet size

The sixth row is the row of packet size. The default packet size of TCP in NS2 is 1000 bytes. But the common packet size of real network is 1500 bytes including a 40 bytes header.

- Packet ID

The last row of the trace file is the row of packet ID. Each packet has its own packet ID in the simulation.

The awk code analyses each line of the trace file and counts the packet sent, received and dropped respectively by judging the action, the source node and the

destination node. TCP packets are separated from ACKs by the packet size. And the packet lost is indicated by packet ID. The expression of the average queue length is:

$$q = a - d - l \qquad 3.2$$

where q is the average queue length, a is the amount of arrived packet, d is the amount of departure packet, and l is the amount of packet dropped.

The simulation time is 100 seconds. However, each simulation model has the period getting stable. The period is called warm up period. To avoid the time of warming up, the data we utilized is from 60 seconds to 100 seconds.

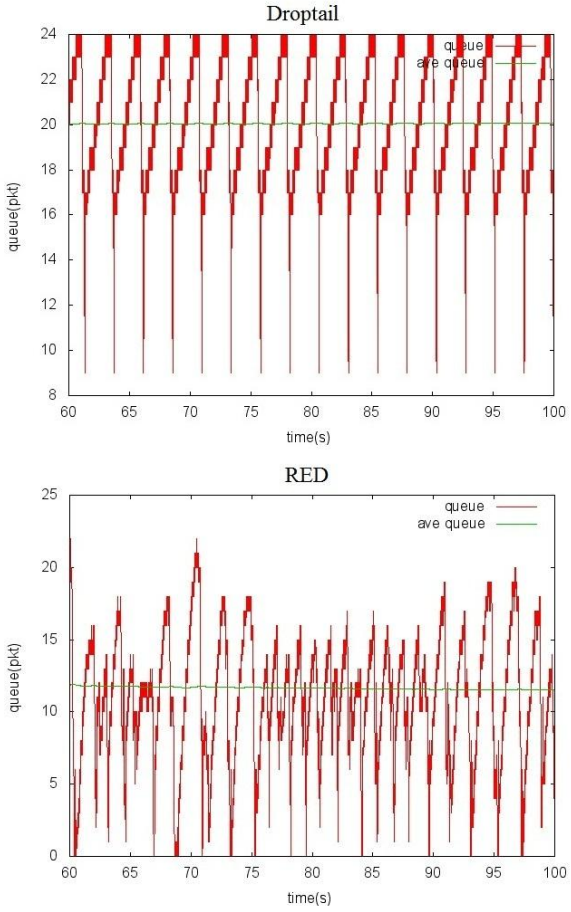

Figure 3.3. *Queue length of Droptail and RED with 2 nodes and no packet loss*

Figure 3.3 shows the queue length of Droptail and RED. The graphs are obtained by gnuplot. The red lines are the instant queue length and the green lines are the average queue length. However, the average queue length of RED is the real average queue length, not the one to calculate the dropping probability.

3.3 802.11 model

3.3.1 Topology

The 802.11 model simulates the WIFI network. All of the nodes in the model are wireless nodes. It is to obtain the relationship between the amount of nodes and the packet loss probability in a wireless network [14] [27].

The topology of 802.11 model is as following:

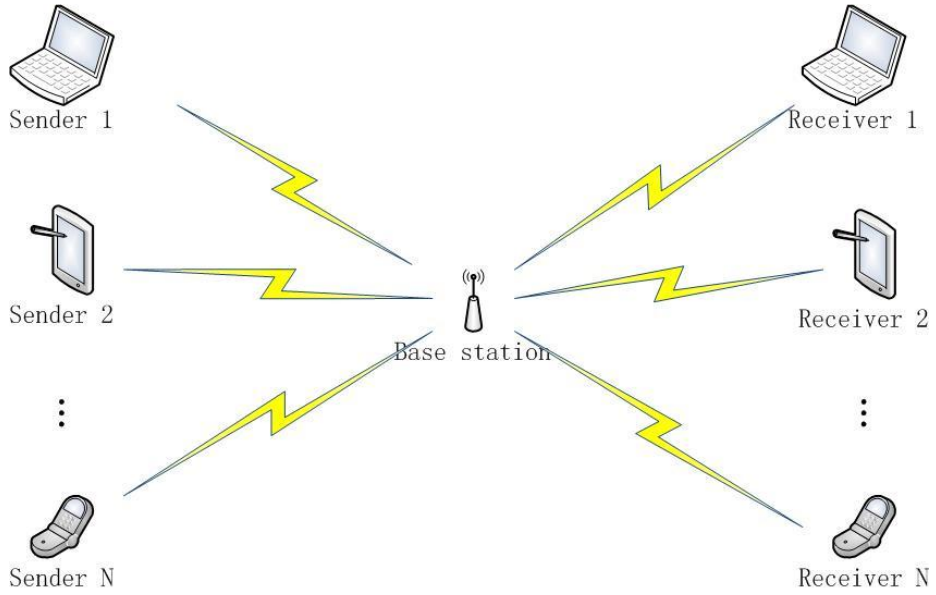

Figure 3.4. *Topology of 802.11 model*

The network contains one base station and several senders and receivers. The number of senders and receivers can be set to 5, 10, 20, or 40. The base station and the receivers belong to one domain, i.e. a WIFI network. Each sender transmits data to the receiver have the same number to it through the base station.

The topology is quite similar to the wired network, but the codes of NS2 are different. Several parameters of wireless network such as MAC layer type and Physical layer type should be set. The base station can only control the nodes within the reception distance, which is determined by the transmission power and reception threshold.

3.3.2 Physical layer parameters

The physical layer parameters determine the distance of transmission. Some important parameters are:

- CSThresh_

CSThresh_ is the carrier sense threshold of the model [19]. In 802.11 model, all nodes utilize CSMA/CA to sense the channel before transmitting. The way to determine if another node is in the domain is to sense the signal strength. So CSThresh_ is the threshold of the signal strength for determining the existing of another node. If the signal strength is less than CSThresh_, the node has no influence to the source node. CSThresh_ is generally 2.2 times larger than RXThresh_.

- RXThresh_

RXThresh_ refers to the reception threshold. If the signal strength is less than RXThresh_, the data cannot be received by the receiver.

- bandwidth_

bandwidth_ is the signal bandwidth.

- Pt_

Pt_ is the transmitting power of the sender. But it will decrease during the transmission.

- freq_

freq_ is the frequency of the signal.

3.3.3 Domain address

Domain addresses are added to the WIFI network [12]. As mentioned before, there are two domains in the simulation. One contains all senders, which can be called domain 0; the other consists of the base station and receivers, is called domain 1. Each domain includes only one cluster. So domain 0 contains cluster 0.0, and domain 1 contains cluster 1.0. Each cluster consists of several nodes. The structure of domain addresses can be presented as below:

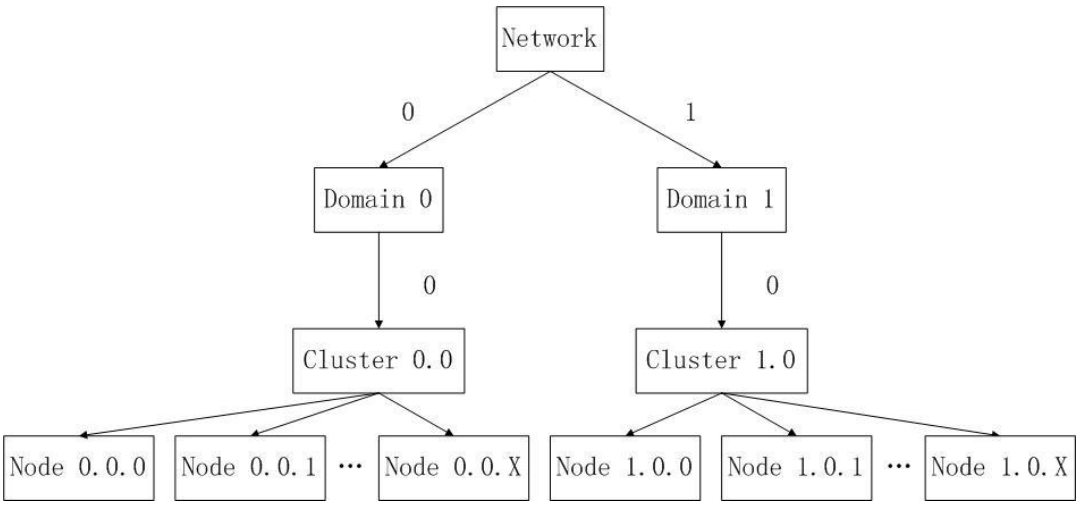

Figure 3.5. *Structure of domain addresses*

The address is 0.0.X for the first domain, and 1.0.X for the second domain. Domain addresses are written in the trace file of the simulation instead of node addresses.

3.3.4 Probability of packet loss

The probability of packet loss is caused by CSMA/CA with exponential backoff. It is calculated by an awk file. The code counts the packets sent and dropped by analysing the trace file. The trace file of wireless network is different from the trace file of wired network [18]:

```
s 90.327930807 _10_ AGT  --- 5836 tcp 1500 [0 0 0 0] ------- [9:0 4194314:0
32 0] [201 0] 0 0
r 90.327930807 _10_ RTR  --- 5836 tcp 1500 [0 0 0 0] ------- [9:0 4194314:0
32 0] [201 0] 0 0
f 90.327930807 _10_ RTR  --- 5836 tcp 1520 [0 0 0 0] ------- [9:0 4194314:0
32 4194304] [201 0] 0 0
```

We use row 1, 2, 4, 7, and 8 in the awk code:

- Action

Action is in the first row. It contains four symbols: ‘s’ (send), ‘r’ (receive), ‘d’ (drop) and ‘f’ (forward).

- Time

The second row is the time that the action happens.

- Trace name

Trace name is in the fourth row. It is the name of the trace. In wireless model, there are several tracing types, such as AGT (agent trace), RTR (router trace) and MAC (MAC trace). One can turn different traces on or off by set those parameters before simulation:

```
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
```

- Packet type

Packet type in row 7 indicates the type of the packet. In this model, it can be TCP, ACK or message.

- Packet size

Same to wired network, the default packet size in row 8 is 1000 bytes. Here we change it to 1500 bytes including a 40 bytes header.

Because the model may contain 40 nodes, it needs a long time to warm up and make the system stable. So the simulation time is 700 seconds, and we only take the information of the last 200 seconds as data.

The details of parameter and the results of the two models are showed in next chapter.

4. RESULTS AND DISCUSSION

4.1 Results of the abstract model

4.1.1 Parameters

The parameters of the abstract model are as the following:

- probability of packet loss: set as 0, 0.01, 0.02, 0.05 or 0.1;
- number of senders: set as 5, 10, 20, or 40;
- bandwidth between senders and routers: 10 Mb;
- bandwidth between the two routers: 1.5 Mb;
- maximum buffer size: set as 25 or 50;
- delay between senders and routers: 3 ms;
- delay between the two routers: 20 ms;
- max bound on TCP agent window size: 16;
- TCP packet size: 1460 bytes;
- RED min threshold: 10 packets for the buffer size of 25 packets; 20 packets for the buffer size of 50 packets;
- RED max threshold: equals to the maximum buffer size;
- RED linterm_: 1;
- RED q_weight_: 0.002;
- simulation time: the simulation runs for 100 seconds, and the data is obtained in the last 40 seconds.

The RED dropping probability is showed in figure 4.1 with different buffer size. The minimum threshold is 10 packets, and the maximum threshold equals to the buffer size as 25 packets. The dropping probability starts from 0 from 10 packets, and becomes 1 when the average queue size reaches the buffer size. If the buffer size changes from 25 packets to 50 packets, the minimum threshold is 20 packets and the maximum threshold still equals to the buffer size as 50 packets. The dropping probability varies from 0 to 1 between the two thresholds.

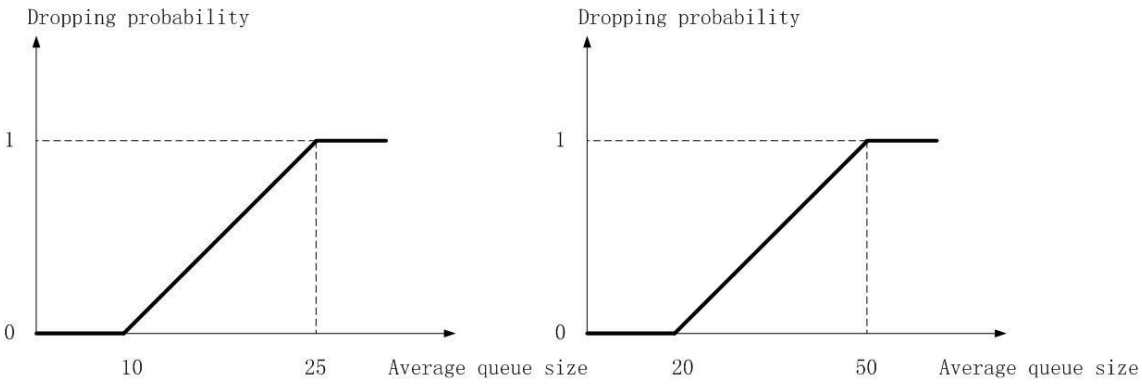

Figure 4.1. *Dropping probability of RED in abstract model*

4.1.2 Graphs of the abstract model and description

With the parameters given above, we got the following graphs about the throughput and average queue size of RED and Droptail. Although packet delay is a reason of packet loss, we do not consider it in our simulation, because comparing to the number of packets dropped due to queue management, the number of packets lost by packet delay is quite small. On the two figures below, the average queue size is the average queue length of the buffer in Router 1:

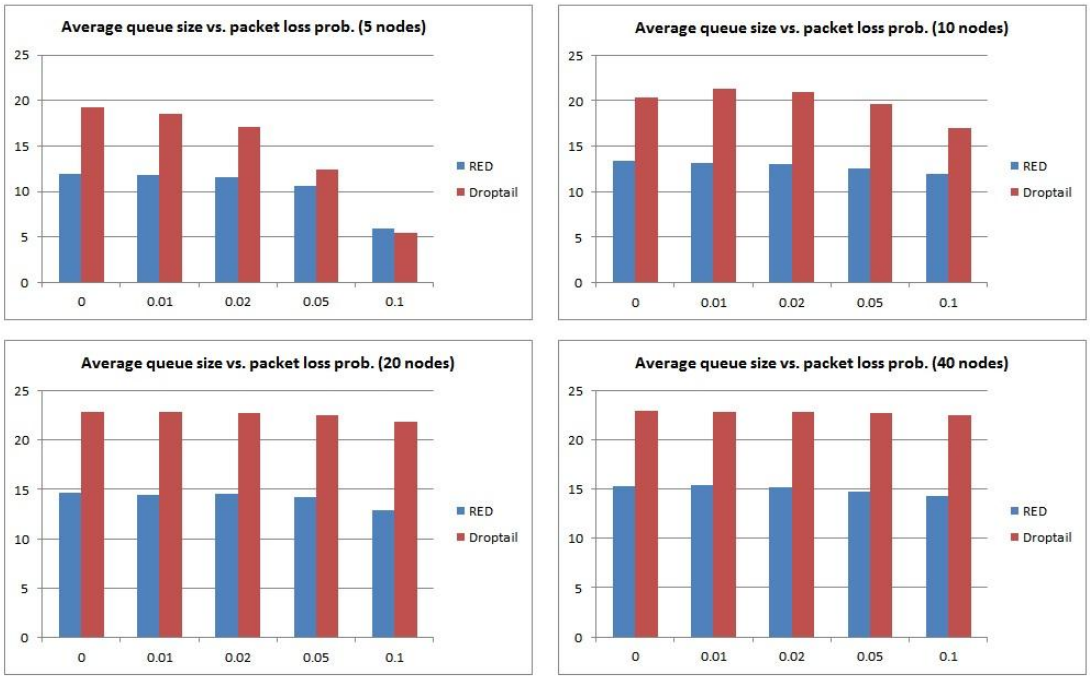

Figure 4.2. *Average queue size vs. packet loss probability with 25 packets buffer size*

Figure 4.2 presents the average queue size vs. packet loss probability with 25 packet

buffer size. The packet loss probability means the external sources of losses that are brought by the error model, not by Droptail or RED. From the pictures, we know that the average queue size of Droptail is large than RED, except the one in the first graph with a packet loss probability of 0.1. The reason is that the probability of packet loss is huge and the amount of nodes is limited. So both Droptail and RED have very low link utilization.

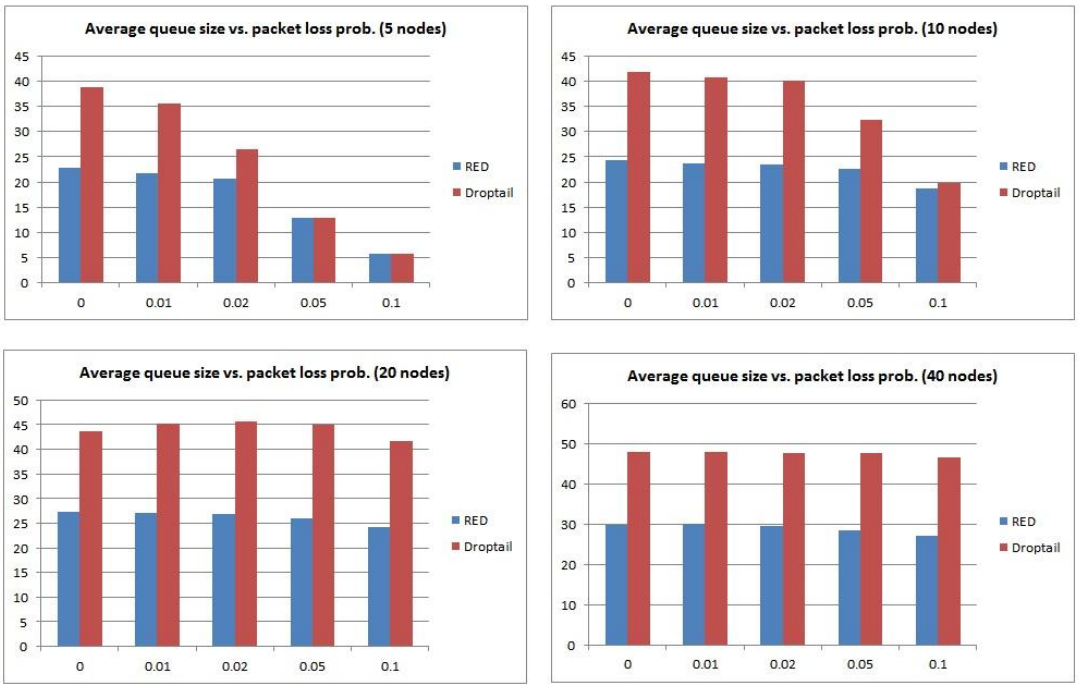

Figure 4.3. *Average queue size vs. packet loss probability with 50 packets buffer size*

Figure 4.3 shows the average queue size vs. packet loss probability with 50 packets buffer size. Same to figure 4.2, it is obvious that the average queue of Droptail is larger than it of RED in most of the cases. When the amount of nodes is small, the average queue length of RED and Droptail are approximate in high packet loss probability. However, as the amount of nodes grows up, the average queue size of Droptail increases faster than it of RED. The reason is that RED controls the queue length actively. The packet may be dropped before the buffer is full. So the sender slowly starts the transmitting rate before the queue size getting larger.

The throughput is obtained by tracing the link between the first sender and its receiver. All data of the throughput is presented with a confidence interval of 95%:

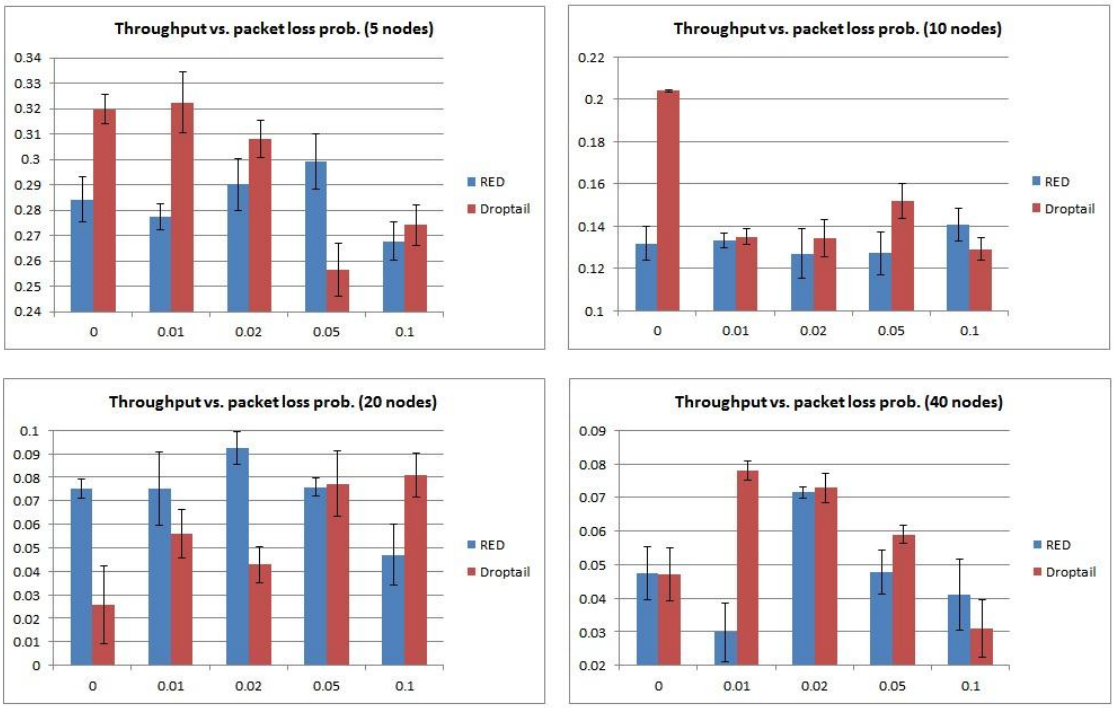

Figure 4.4. *Throughput vs. packet loss probability with 25 packets buffer size*

The figure above shows the throughput vs. packet loss probability with 25 packets buffer size. We cannot see superiority of RED in this figure, even we have added confidence interval.

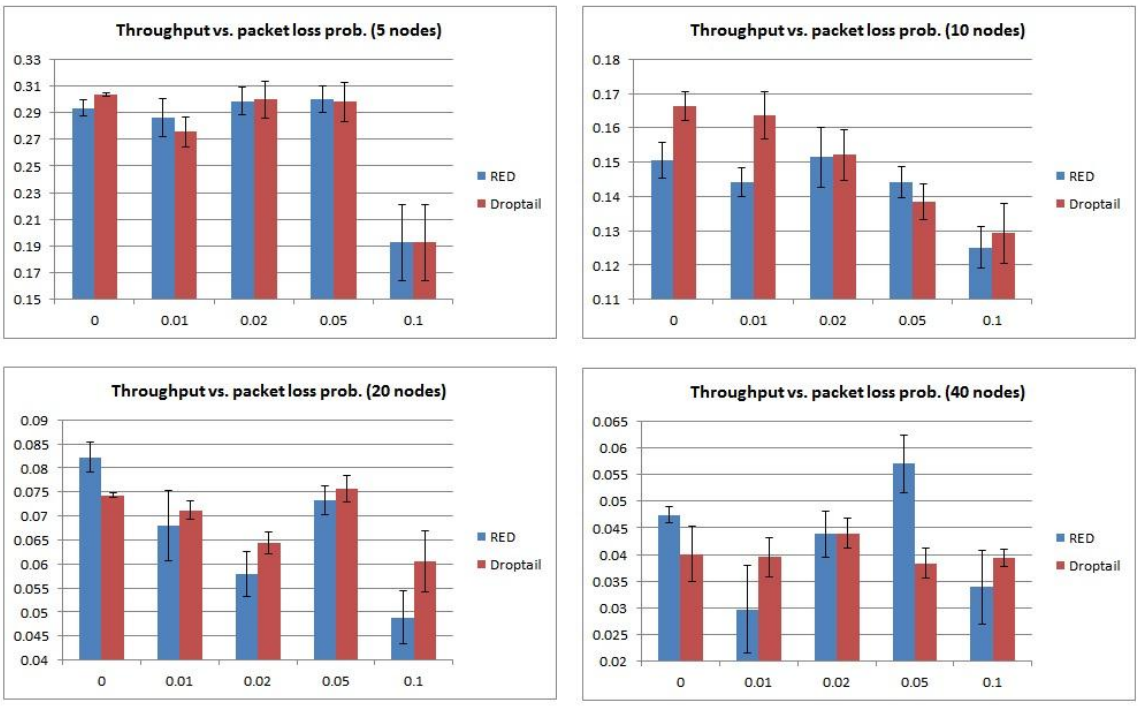

Figure 4.5. *Throughput vs. packet loss probability with 50 packets buffer size*

Figure 4.5 is similar to figure 4.4 but the buffer size is changed to 50 packets. From the figures above, we know that in different packet loss probabilities, there is no superiority between RED and Droptail. Consider about the confidence interval, the differences between RED and Droptail in throughput are not huge, even the maximum buffer size is changed from 25 packets to 50 packets.

Although RED gets less queue size than Droptail, the throughputs of the two algorithms are approximate. Despite RED sometimes has better performance than Droptail and vice versa, we cannot say that RED has the superiority, because the amount of nodes in a real wireless network changes every moment. It is not needed to use two different queue management algorithms in one network. Compared to Droptail, RED is more complicated. So it is not necessary to use RED in wireless network.

4.2 Results of the 802.11 model

4.2.1 Parameters

We choose TwoRayGround as the wireless channel of the model. The parameters of the simulation are showed below:

- CSMA/CA threshold: 7.5195e-12 W;
- reception threshold: 1.76149e-10 W;
- bandwidth: 54 Mbps;
- transmission power: 0.28183815 W;
- the parameters of TCP and RED are the same to the abstract model.

The simulation runs for 700 seconds. And the data is obtained from 500 seconds to 700 seconds to remove the influence of warm up. The packet loss probability is the result of CSMA/CA with exponential backoff.

4.2.2 Graphs of 802.11 model and description

The results are presented below:

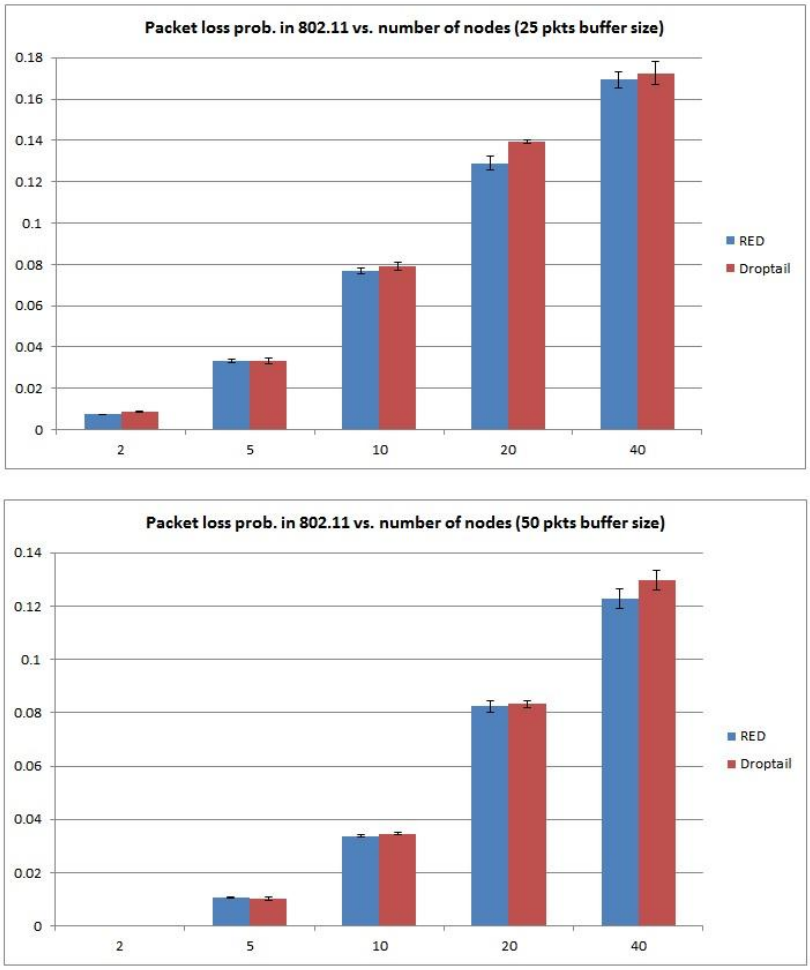

Figure 4.6. *Probability of packet loss vs. number of nodes*

As showed on the two figures, the packet loss probability of RED and Droptail are approximate with a certain amount of nodes. From the abstract model, we know that the throughputs of RED and Droptail under the same packet loss probability are approximate. Then combine the two models, we can get the relationship between buffer size and amount of nodes, and throughput and amount of modes.

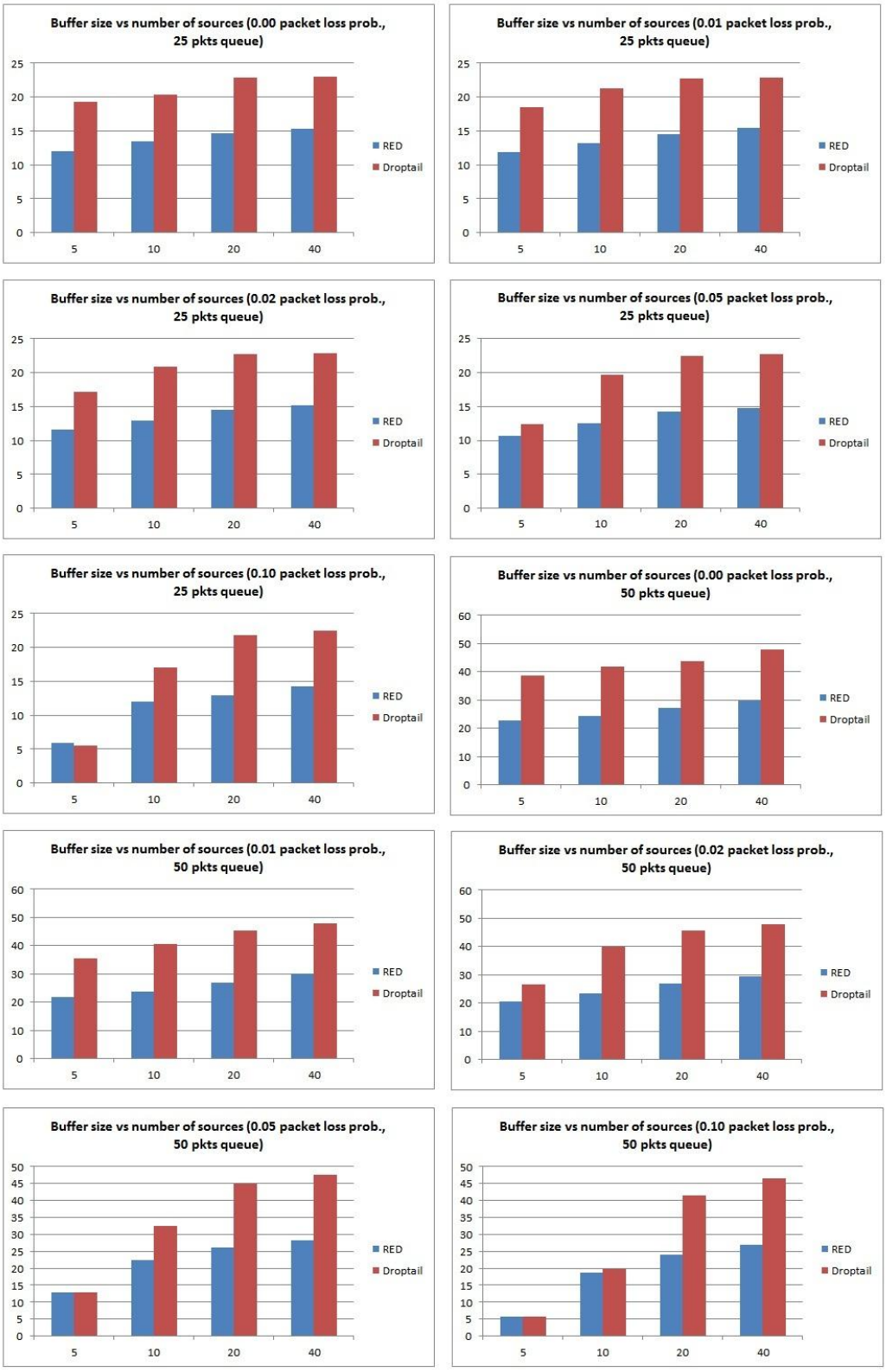

Figure 4.7. *Buffer size vs. number of nodes*

In figure 4.7, we obtain the similar results to figure 4.2 and 4.3. The average queue length of RED is smaller than the average queue length of Droptail. That means the delay of RED is shorter than the delay of Droptail.

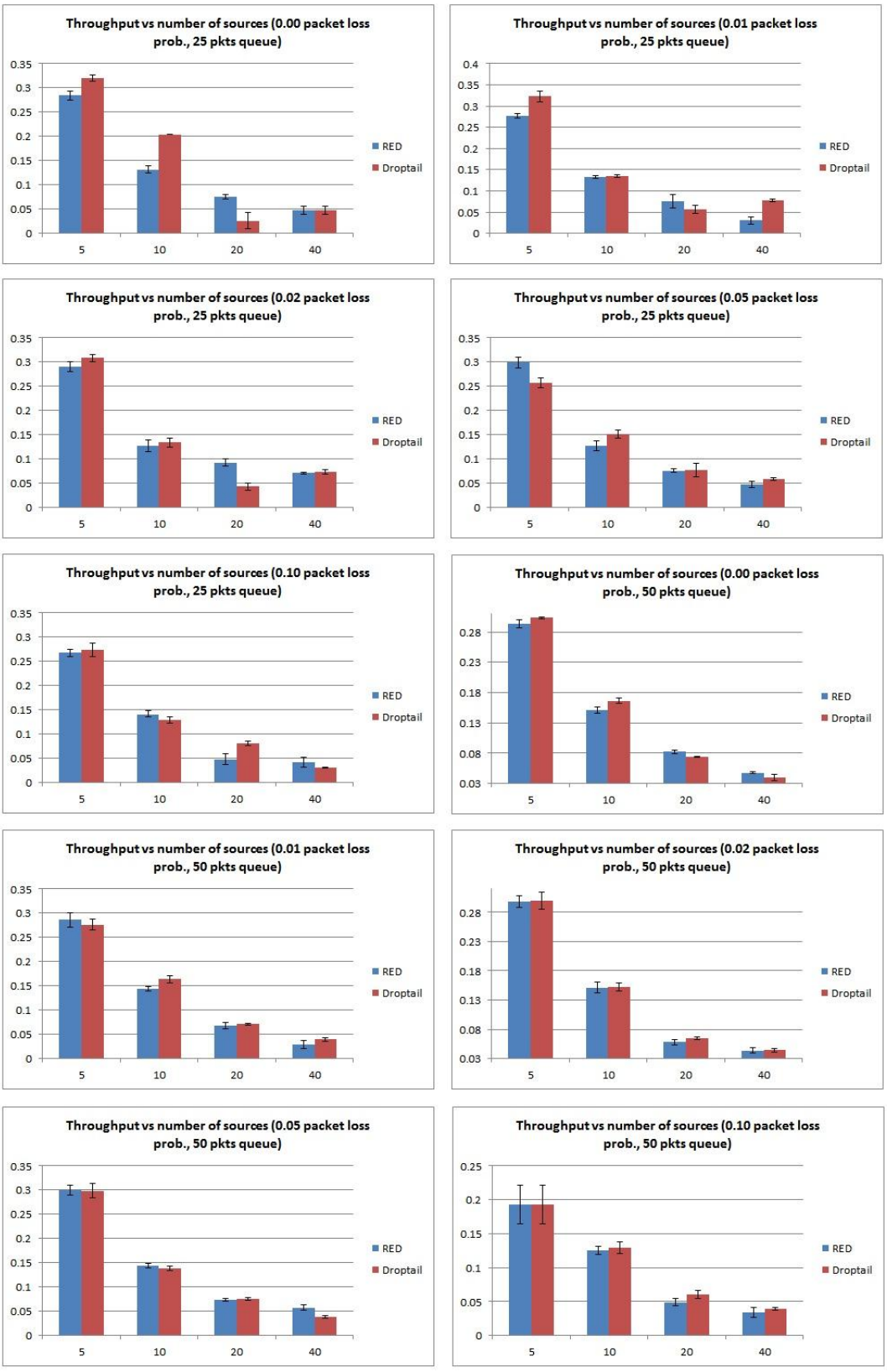

Figure 4.8. *Throughput vs. number of nodes*

In figure 4.8, we know that RED does not show the superiority in most of the cases. The result is definitely same to the results of figure 4.4 and figure 4.5. Due to this result, and the fact that RED is complicated than Droptail, we do not recommended RED in wireless network, although it has a shorter packet delay than Droptail.

5. CONCLUSIONS

In this thesis, we compared the two queue management algorithms: RED and Droptail in wireless network, i.e. 802.11. We analysed the differences between the two algorithms and reviewed the performances of them in wired network. RED is more complicated than Droptail, but has better performance in wired network.

However, wireless network is quite different from wired network. It has more nodes for one base station than that for wired network. And it has external source of losses that may influence the network performance. Due to those reasons, we made two simulation models to test the performances of RED and Droptail in 802.11. The abstract model uses wired network but with external packet loss probability. With this model we first proved that the average queue size of RED is short than if of Droptail. Small queue size represents short delay, so if RED gets the same throughput to Droptail, it is still recommended. But then we found that the throughput of RED has no superiority compared to Droptail when same external source of losses are added. In the 802.11 model, it proved that the packet loss probabilities of RED and Droptail for a certain amount of nodes are approximate. Combining the two simulations, we know that RED is not better than Droptail when there is external source of losses.

The further work could be finding the queue management algorithm that is available for wireless network with external source of losses.

6. APPENDIX

The abstract model for Droptail:

```
#===================================================================
# Define options
#===================================================================
set ns [new Simulator]
set nf [open out1.nam w]
$ns namtrace-all $nf
set tracefd       [open out.tr w]
$ns trace-all $tracefd
#===================================================================
# packetloss
#===================================================================
set loss_module [new ErrorModel]
$loss_module set rate_ 0.01
$loss_module unit pkt
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
#===================================================================
# set nodes
#===================================================================
set number 20
for {set i 0} {$i < $number } {incr i} {
      set n($i) [$ns node]
}
for {set i 0} {$i < $number } {incr i} {
      set s($i) [$ns node]
}
set r1 [$ns node]
set r2 [$ns node]
for {set i 0} {$i < $number } {incr i} {
   $ns duplex-link $n($i) $r1 10Mb 3ms DropTail
}
for {set i 0} {$i < $number } {incr i} {
   $ns duplex-link $s($i) $r2 10Mb 3ms DropTail
}
$ns duplex-link $r1 $r2 1.5Mb 20ms DropTail
```

```
$ns queue-limit $r1 $r2 50
$ns queue-limit $r2 $r1 50
$ns link-lossmodel $loss_module $r1 $r2
#=============================================================
# set TCP
#=============================================================
for {set i 0} { $i < $number } {incr i} {
       set tcp($i) [new Agent/TCP/Reno]
      $tcp($i) set class_ $i
      $tcp($i) set window_ 16
      $tcp($i) set packetSize_ 1460
      $ns attach-agent $n($i) $tcp($i)
}
for { set i 0 } { $i < $number} { incr i } {
       set sink($i) [new Agent/TCPSink]
       $ns attach-agent $s($i) $sink($i)
       $ns connect $tcp($i) $sink($i)
       set ftp($i) [$tcp($i) attach-source FTP]
}
#=============================================================
# Tracing the ave_throuput and throughput of tcp1 in Mbps
#=============================================================
set ftcp [open TCP_Reno.tr w]
set hold 0
puts $ftcp "a 0 0"
proc record {} {
       global sink hold ns ftcp bth
       set intval 0.2
       set bt [$sink(0) set bytes_]
       set now [$ns now]
       if { $now > 0 } {
            puts $ftcp "a $now [expr ($bt)*8/(($now*1.0)*1000000)]"
       }
       puts $ftcp "t $now [expr ($bt-$hold)*8/($intval*1000000)]"
       set hold $bt
       $ns at [expr $now+$intval] "record"
  }
#=============================================================
# set time
#=============================================================
for { set i 0 } { $i < $number } { incr i } {
       $ns at 0.0 "$ftp($i) start"
}
$ns at 0.0 "record"
```

```
$ns at 100.0 "finish"
#=============================================================
# define finish
#=============================================================
proc finish {} {
        global nf tracefd ftcp ns
        set awkCode1 {
           {
                if ($1 == "a" && $2 > 60) {
                   print $2,$3 >> "temp1.tr"
                }
                else if ($1 == "t" && $2 > 60) {
                   print $2,$3 >> "temp2.tr"
                }
           }
        }
        set t [ open temp.tr w]
        if { [info exists ftcp] } {
       close $ftcp
        }
        exec rm -f temp1.tr temp2.tr
        exec touch temp2.tr temp1.tr
        exec awk $awkCode1 TCP_Reno.tr
        puts $t \"ave_throughput
        exec cat temp1.tr >@ $t
        puts $t \n\"throughput
        exec cat temp2.tr >@ $t
        close $nf
        close $tracefd
        close $t
        exec nam out1.nam &
        exec xgraph -bb -tk -x time -y throughput(Mbps) temp.tr &
        exit 0
}
$ns run
```

The abstract model for RED:

```
#=============================================================
# Define options
#=============================================================
set ns [new Simulator]
set nf [open out1.nam w]
$ns namtrace-all $nf
```

```
set tracefd [open out.tr w]
$ns trace-all $tracefd
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set thresh_ 20
Queue/RED set maxthresh_ 50
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.002
Queue/RED set drop_tail_ true
#=============================================================
# packetloss
#=============================================================
set loss_module [new ErrorModel]
$loss_module set rate_ 0.00
$loss_module unit pkt
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
#=============================================================
# set nodes
#=============================================================
set number 5
for {set i 0} {$i < $number } {incr i} {
      set n($i) [$ns node]
}
for {set i 0} {$i < $number } {incr i} {
      set s($i) [$ns node]
}
set r1 [$ns node]
set r2 [$ns node]
for {set i 0} {$i < $number } {incr i} {
   $ns duplex-link $n($i) $r1 10Mb 3ms DropTail
}
for {set i 0} {$i < $number } {incr i} {
   $ns duplex-link $s($i) $r2 10Mb 3ms DropTail
}
$ns duplex-link $r1 $r2 1.5Mb 20ms RED
$ns queue-limit $r1 $r2 50
$ns queue-limit $r2 $r1 50
$ns link-lossmodel $loss_module $r1 $r2
#=============================================================
# set TCP
#=============================================================
for {set i 0} { $i < $number } {incr i} {
      set tcp($i) [new Agent/TCP/Reno]
```

```
        $tcp($i) set class_ $i
        $tcp($i) set window_ 16
        $tcp($i) set packetSize_ 1460
        $ns attach-agent $n($i) $tcp($i)
}
for { set i 0 } { $i < $number} { incr i } {
        set sink($i) [new Agent/TCPSink]
        $ns attach-agent $s($i) $sink($i)
        $ns connect $tcp($i) $sink($i)
        set ftp($i) [$tcp($i) attach-source FTP]
}
#=================================================================
# Tracing the ave_throuput and throughput of tcp1 in Mbps
#=================================================================
set ftcp [open TCP_Reno.tr w]
set hold 0
puts $ftcp "a 0 0"
proc record {} {
        global sink hold ns ftcp
        set intval 0.2
        set bt [$sink(0) set bytes_]
        set now [$ns now]
        if { $now > 0 } {
             puts $ftcp "a $now [expr ($bt)*8/(($now*1.0)*1000000)]"
        }
        puts $ftcp "t $now [expr ($bt-$hold)*8/($intval*1000000)]"
        set hold $bt
        $ns at [expr $now+$intval] "record"
  }
#=================================================================
# set time
#=================================================================
for { set i 0 } { $i < $number } { incr i } {
        $ns at 0.0 "$ftp($i) start"
}
$ns at 0.0 "record"
$ns at 100.0 "finish"
#=================================================================
# define finish
#=================================================================
proc finish {} {
        global nf tracefd ftcp ns
        set awkCode1 {
           {
```

```
                if ($1 == "a" && $2 > 60) {
                    print $2,$3 >> "temp1.tr"
                }
                else if ($1 == "t" && $2 > 60) {
                    print $2,$3 >> "temp2.tr"
                }
            }
        }
        set t [ open temp.tr w]
        if { [info exists ftcp] } {
        close $ftcp
        }
        exec rm -f temp1.tr temp2.tr
        exec touch temp2.tr temp1.tr
        exec awk $awkCode1 TCP_Reno.tr
        puts $t \"ave_throughput
        exec cat temp1.tr >@ $t
        puts $t \n\"throughput
        exec cat temp2.tr >@ $t
        close $nf
        close $tracefd
        close $t
        exec nam out1.nam &
        exec xgraph -bb -tk -x time -y throughput(Mbps) temp.tr &
        exit 0
}
$ns run
```

The awk code for abstract model to get the average queue size:

```
BEGIN {
loss=0;
arr=0;
dep=0;
#arrive time
arrtime=0;
id=0;
#to get ave_queue
loop=0;
sumqueue[0]=0;
#to get the loop when time > 60
time60=0;
set=0;
}
```

```
{
   loop=loop+1;
   action=$1;
   time[loop]=$2;
   from=$3;
   to=$4;
   pktsize=$6;
   packet_id=$12;
   nothing=0;
if(time[loop] >= 60 && set==0){
   set=1;
   time60=loop;
}
if(action == "+" && from == "40" && to == "41" && pktsize == 1500){
     arr=arr+1;
     arrtime=time[loop];
     id=packet_id;
     qsize[loop]=arr-dep-loss;
     sumqueue[loop]=sumqueue[loop-1]+qsize[loop];
     aveq[loop]=sumqueue[loop]/(loop*1.0);
     nothing=1;
}
if(action == "-" && from == "40" && to == "41" && pktsize == 1500){
     dep=dep+1;
     if (time[loop] == arrtime) {
        loop=loop-1;
        qsize[loop]=qsize[loop]-1;
        sumqueue[loop]=sumqueue[loop]-1;
        aveq[loop]=sumqueue[loop]/(loop*1.0);
        arrtime=0;
     }
     else{
        qsize[loop]=arr-dep-loss;
        sumqueue[loop]=sumqueue[loop-1]+qsize[loop];
        aveq[loop]=sumqueue[loop]/(loop*1.0);
     }
   nothing=1;
}
if(action == "d" && from == "40" && to == "41" && pktsize == 1500){
   if(arrtime == time[loop] && id == packet_id){
     loop=loop-1;
     loss=loss+1;
     qsize[loop]=qsize[loop]-1;
     sumqueue[loop]=sumqueue[loop]-1;
```

```
    aveq[loop]=sumqueue[loop]/(loop*1.0);
    arrtime=0;
   }
   else{
     loop=loop-1;
   }
nothing=1;
}
if(nothing==0){
    if(time60==time[loop]){
       time60=0;
       set=0;
    }
   loop=loop-1;
}
}
END {
for(i=time60;i <= loop; i++){
    printf("%f\t%f\t%f\n",time[i],qsize[i], aveq[i]);
}
}
```

The 802.11 model for Droptail:

```
#=================================================================
# Define options
#=================================================================
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
set val(rp)         DSDV
set val(ifqlen)    50
#Phy/WirelessPhy set CPThresh_ 285.759
Phy/WirelessPhy set CSThresh_ 7.5195e-12; # 660m,the carrier sensing
# threshold
Phy/WirelessPhy set RXThresh_     1.76149e-10 ; # 300m,the reception threshold
Phy/WirelessPhy set bandwidth_ 54e6
Phy/WirelessPhy set Pt_ 0.28183815
Phy/WirelessPhy set freq_ 914e+6
set ns [new Simulator]
```

```
set tracefd    [open WiFi.tr w]
set nf [open WiFi.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $nf 500 500
set topo [new Topography]
$topo load_flatgrid 500 500
set number 2
set chan [new $val(chan)]
set god [create-god    [expr 2*$number+1]]
$ns node-config                 -adhocRouting $val(rp) \
                                    -llType $val(ll) \
                                    -macType $val(mac) \
                                    -ifqType $val(ifq) \
                                    -ifqLen $val(ifqlen) \
                                    -antType $val(ant) \
                                    -propType $val(prop) \
                                    -phyType $val(netif) \
                                    -channel $chan \
                                    -agentTrace ON \
                                    -routerTrace ON \
                                    -macTrace OFF \
                                    -addressType hierarchical \
                                    -topoInstance $topo \
                                    -movementTrace OFF
                                    # -channelType $val(chan) \
#=============================================================
# set nodes
#=============================================================
AddrParams set domain_num_ 2          ;# number of domains
lappend cluster_num 1 1             ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel [expr $number] [expr $number+1]
AddrParams set nodes_num_ $eilastlevel ;# number of nodes in cluster
set r [ $ns node 1.0.0]
$r random-motion 0
$r set X_ 250.0
$r set Y_ 250.0
$r set Z_ 0.0
for {set i 0} {$i < $number } {incr i} {
      set n($i) [ $ns node 0.0.[expr $i]]
      $n($i) random-motion 0
      $n($i) set X_ 50.0
      # to get connected with r, 30<y<470
      $n($i) set Y_ [expr 30.0+440.0/($number-1)*$i]
```

```
        $n($i) set Z_ 0.0
}
for {set i 0} {$i < $number } {incr i} {
        set s($i) [ $ns node 1.0.[expr $i+1]]
        $s($i) base-station [AddrParams addr2id [$r node-addr]]
        # provide each sink with hier address of its base-station
        $s($i) random-motion 0
        $s($i) set X_ 450.0
        $s($i) set Y_ [expr 30.0+440.0/($number-1)*$i]
        $s($i) set Z_ 0.0
}
for {set i 0} {$i < $number } {incr i} {
$ns initial_node_pos $n($i) 20
}
for {set i 0} {$i < $number } {incr i} {
$ns initial_node_pos $s($i) 20
}
$ns initial_node_pos $r 50
for {set i 1} {$i <= [expr $number*2] } {incr i} {
   $god set-dist 0 [expr $i] 1
}
for {set i 0} {$i < $number } {incr i} {
   $god set-dist [expr $i+1] [expr $i+$number+1] 2
}
#=============================================================
# set TCP
#=============================================================
for {set i 0} { $i < $number } {incr i} {
          set tcp($i) [new Agent/TCP/Reno]
      $tcp($i) set window_ 16
      $tcp($i) set packetSize_ 1460
      $ns attach-agent $n($i) $tcp($i)
}
for { set i 0 } { $i < $number} { incr i } {
        set sink($i) [new Agent/TCPSink]
        $ns attach-agent $s($i) $sink($i)
        $ns connect $tcp($i) $sink($i)
        set ftp($i) [$tcp($i) attach-source FTP]
}
#=============================================================
# set time
#=============================================================
for { set i 0 } { $i < $number } { incr i } {
        $ns at 0.0 "$ftp($i) start"
```

```
}
$ns at 700.0 "finish"
#=============================================================
# define finish
#=============================================================
proc finish {} {
        global nf tracefd ns
        close $nf
        close $tracefd
        exec nam WiFi.nam &
        exit 0
}
$ns run
```

The 802.11 model for RED:

```
#=============================================================
# Define options
#=============================================================
Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set thresh_ 20
Queue/RED set maxthresh_ 50
Queue/RED set linterm_ 1
Queue/RED set q_weight_ 0.002
Queue/RED set drop_tail_ true
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)         Mac/802_11
set val(ifq)        Queue/RED
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
set val(rp)         DSDV
set val(ifqlen)    50
#Phy/WirelessPhy set CPThresh_ 285.759
Phy/WirelessPhy set CSThresh_ 7.5195e-12; # 660m,the carrier sensing
# threshold
Phy/WirelessPhy set RXThresh_     1.76149e-10 ; # 300m,the reception threshold
Phy/WirelessPhy set bandwidth_ 54e6
Phy/WirelessPhy set Pt_ 0.28183815
Phy/WirelessPhy set freq_ 914e+6
set ns [new Simulator]
set tracefd    [open WiFi.tr w]
```

```
set nf [open WiFi.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $nf 500 500
set topo [new Topography]
$topo load_flatgrid 500 500
set number 2
set chan [new $val(chan)]
set god [create-god    [expr 2*$number+1]]
$ns node-config                  -adhocRouting $val(rp) \
                                    -llType $val(ll) \
                                    -macType $val(mac) \
                                    -ifqType $val(ifq) \
                                    -ifqLen $val(ifqlen) \
                                    -antType $val(ant) \
                                    -propType $val(prop) \
                                    -phyType $val(netif) \
                                    -channel $chan \
                                    -agentTrace ON \
                                    -routerTrace ON \
                                    -macTrace OFF \
                                    -addressType hierarchical \
                                    -topoInstance $topo \
                                    -movementTrace OFF
                                    # -channelType $val(chan) \
#=============================================================
# set nodes
#=============================================================
AddrParams set domain_num_ 2          ;# number of domains
lappend cluster_num 1 1               ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel [expr $number] [expr $number+1]
AddrParams set nodes_num_ $eilastlevel ;# number of nodes in cluster
set r [ $ns node 1.0.0]
$r random-motion 0
$r set X_ 250.0
$r set Y_ 250.0
$r set Z_ 0.0
for {set i 0} {$i < $number } {incr i} {
      set n($i) [ $ns node 0.0.[expr $i]]
      $n($i) random-motion 0
      $n($i) set X_ 50.0
      # to get connected with r, 30<y<470
      $n($i) set Y_ [expr 30.0+440.0/($number-1)*$i]
      $n($i) set Z_ 0.0
```

```
}
for {set i 0} {$i < $number } {incr i} {
       set s($i) [ $ns node 1.0.[expr $i+1]]
       $s($i) base-station [AddrParams addr2id [$r node-addr]]
       # provide each sink with hier address of its base-station
       $s($i) random-motion 0
       $s($i) set X_ 450.0
       $s($i) set Y_ [expr 30.0+440.0/($number-1)*$i]
       $s($i) set Z_ 0.0
}
for {set i 0} {$i < $number } {incr i} {
$ns initial_node_pos $n($i) 20
}
for {set i 0} {$i < $number } {incr i} {
$ns initial_node_pos $s($i) 20
}
$ns initial_node_pos $r 50
for {set i 1} {$i <= [expr $number*2] } {incr i} {
   $god set-dist 0 [expr $i] 1
}
for {set i 0} {$i < $number } {incr i} {
   $god set-dist [expr $i+1] [expr $i+$number+1] 2
}
#=============================================================
# set TCP
#=============================================================
for {set i 0} { $i < $number } {incr i} {
          set tcp($i) [new Agent/TCP/Reno]
      $tcp($i) set window_ 16
      $tcp($i) set packetSize_ 1460
      $ns attach-agent $n($i) $tcp($i)
}
for { set i 0 } { $i < $number} { incr i } {
       set sink($i) [new Agent/TCPSink]
       $ns attach-agent $s($i) $sink($i)
       $ns connect $tcp($i) $sink($i)
       set ftp($i) [$tcp($i) attach-source FTP]
}
#=============================================================
# set time
#=============================================================
for { set i 0 } { $i < $number } { incr i } {
       $ns at 0.0 "$ftp($i) start"
}
```

```
$ns at 700.0 "finish"
#=============================================================
# define finish
#=============================================================
proc finish {} {
        global nf tracefd ns
        close $nf
        close $tracefd
        exec nam WiFi.nam &
        exit 0
}
$ns run
```

The awk code for 802.11 model to get the packet loss probability:

```
BEGIN {
loss=0;
sent=0;
loop=0;
}
{
   nothing=0;
   loop=loop+1;
   action=$1;
   time[loop]=$2;
   trace=$4;
   protocol=$7;
   psize=$8;
 if(action == "s" && trace == "AGT" && protocol == "tcp" && psize==1500){
   sent=sent+1;
   nothing=1;
 }
 if(action == "D" && protocol == "tcp"){
   loss=loss+1;
   nothing=1;
 }
 if(nothing==1){
    prob[loop]=loss*1.0/(sent*1.0);
 }
 if(nothing==0){
 loop=loop-1;
 }
}
END {
```

```
    for(i=1;i<=loop;i++){
      if (time[i]>=500){
        break;
      }
    }
    for(j=i;j<=loop;j++){
      printf("%f\t%f\n",time[j],prob[j]);
    }
}
```

7. REFERENCE

[1] John Nagle. RFC 896 – Congestion Control in IP/TCP Internetworks. Tools.ietf.org.1984-01-06. http://tools.ietf.org/html/rfc896

[2] Mujdat Soyturk. Design and Analysis of TCP/IP Networks. Gebze Institute of Technology. Spring 2011. Available at: http://www.gyte.edu.tr/hebe/AblDrive/99104018/w/Storage/104_2010_2_574_99104018/Downloads/week3-4activequeuemanagement.pdf

[3] Dmitri Moltchanov. TLT-2727: Queue Management. Tampere University of Technology

[4] Sally Floyd, Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. IEEE/ ACM Transaction on Networking. August 1993

[5] Mats Sagfors, Reiner Ludwig, Michael Meyer, Janne Peisa. Queue Management for TCP Traffic over 3G Links. Wireless Communications and Networking, 2003.

[6] Xinnian Chen. Analysis and Comparison of the NS2 Based Router Algorithms: Droptail and RED. Computer Engineering & Science, vol. 29, No. 6, 2007

[7] Roman Dunaytsev. Congestion Control. Tampere University of Technology. 2010-12-02

[8] Roman Dunaytsev. MAC Techniques. Tampere University of Technology. 2010-10-07

[9] M. Allman, V. Paxson, W. Stevens. RFC 2581 – TCP Congestion Control. Tools.ietf.org. April 1999.

[10] IEEE Standard 802.3-2008". Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. NY 2008. IEEE-SA Standards Board. 67 P. http://standards.ieee.org/getieee802/download/802.3-2008_section1.pdf

[11] Jae Chung, Mark Claypool. NS by Example. Worcester Polytechnic Institute. http://nile.wpi.edu/NS/

[12] Marc Greis. Tutorial for the Network Simulator “ns”. http://www.isi.edu/nsnam/ns/tutorial/index.html

[13] Kevin Fall, Kannan Varadhan. The ns Manual. UC Berkeley, LBL, USC/ ISI, Xerox PARC. 2011-11-04. 126 P. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

[14] NS-2 Tutorial. Demokritos University o Thrace.

http://skontog.gr/work/wlesson.pdf

[15] Arijit Ganguly, Pasi Lassila. A Study of TCP-RED Congestion Control Using NS2. 2001-07-20

[16] General main RED parameters. http://www.cs.technion.ac.il/Courses/Computer-Networks-Lab/projects/spring2003/ns1/Net%20Site/Source/RedParam.htm

[17] Sally Floyd. RED: Discussions of Setting Parameters. Nov 1997. http://www.icir.org/floyd/REDparameters.txt

[18] NS-2 Trace Formats. National Science Foundation. http://nsnam.isi.edu/nsnam/index.php/NS-2_Trace_Formats

[19] Teerawat Issariyakul, Ekram Hossain. Introduction to Network Simulator NS2. Springer. 2008-10-20. 332 P.

[20] Pablo Brenner. A Technical Tutorial on the IEEE 802.11 Protocol. BreezeCOM. 1997. http://www.sss-mag.com/pdf/802_11tut.pdf

[21] 802.11 MAC (Media Access Control). ZyTrax. http://www.zytrax.com/tech/wireless/802_mac.htm

[22] CSMA/CA. http://www.cs.clemson.edu/~westall/851/802.11/802_CSMA_CA.pdf

[23] TCP Congestion Control. http://condor.depaul.edu/jkristof/technotes/congestion.pdf

[24] Shweta Sinha, Andy Ogielski. A TCP Tutorial. http://ssfnet.org/Exchange/tcp/tcpTutorialNotes.html#CC

[25] Ganesh Patil, Sally McClean, Gaurav Raina. Drop Tail and RED Queue Management with Small Buffers: Stability and HOPF Bifurcation. ICTACT Journal on Communication Technology. Jun 2011. Volume 2 Issue 2. http://ictact.in/scripts/Abstract/jct_Spl_Paper_339_344.pdf

[26] Frank Eyermann. Simulating Networks with Network Simulator 2. University of Zurich. 2008-06-03. http://www.aims-conference.org/issnsm-2008/02-ns2-exercises.pdf

[27] Zhibin Wu. NS-2 Wireless Simulation Tutorial. Rutgers University. http://www.winlab.rutgers.edu/~zhibinwu/pdf/ns2tutorial.pdf

[28] Yue Wang. A Tutorial of 802.11 Implementation in NS-2. CUHK. http://www.winlab.rutgers.edu/~zhibinwu/pdf/tr_ns802_11.pdf

[29] Lingyu Lin. Error Model. http://ant.comm.ccu.edu.tw/course/96_Network_Simulation/NS2_class_report_ppts/14_695430014_Error%20Model%20.ppt

[30] Aliff Umair Salleh, Zulkifli Ishak , Norashidah Md. Din, Md Zaini Jamaludin. Trace Analyzer for NS-2. Universiti Tenaga Nasional. http://gdauto.gdut.edu.cn/lab1/lwfy_files/Trace%20Analyzer%20for%20NS-2.pdf

[31] NS-2 Software and Simulation Results.
http://www.cs.ucy.ac.cy/networksgroup/ns2-labs/introduction/performance.pdf
[32] Active Queue Management: Random Early Detection/ Drop.
http://www.mathcs.emory.edu/~cheung/Courses/558/Syllabus/08-RED/RED.html